NASA/CR-1999-209713
ICASE Report No. 99-42

Statecharts via Process Algebra

Gerald Lüttgen
ICASE, Hampton, Virginia

Michael von der Beeck
Munich University of Technology, München, Germany

Rance Cleaveland
State University of New York at Stony Brook, Stony Brook, New York

October 1999

NASA/CR-1999-209713
ICASE Report No. 99-42

Statecharts via Process Algebra

Gerald Lüttgen
ICASE, Hampton, Virginia

Michael von der Beeck
Munich University of Technology, München, Germany

Rance Cleaveland
State University of New York at Stony Brook, Stony Brook, New York

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

October 1999

STATECHARTS VIA PROCESS ALGEBRA*

GERALD LÜTTGEN†, MICHAEL VON DER BEECK‡, AND RANCE CLEAVELAND§

Abstract. *Statecharts* is a visual language for specifying the behavior of *reactive systems*. The language extends *finite-state machines* with concepts of *hierarchy*, *concurrency*, and *priority*. Despite its popularity as a design notation for *embedded systems*, precisely defining its semantics has proved extremely challenging. In this paper, a simple *process algebra*, called *Statecharts Process Language* (SPL), is presented, which is expressive enough for encoding Statecharts in a structure-preserving and semantics-preserving manner. It is established that the behavioral relation *bisimulation*, when applied to SPL, preserves Statecharts semantics.

Key words. bisimulation, compositionality, operational semantics, process algebra, Statecharts

Subject classification. Computer Science

1. Introduction. *Statecharts* is a visual language for specifying the behavior of *reactive systems* [7]. The language extends the traditional notation of *finite-state machines* with concepts of (i) *hierarchy*, so that one may speak of a state as having sub-states, (ii) *concurrency*, thereby allowing the definition of systems having simultaneously active subsystems, and (iii) *priority*, so that one may express that certain system activities have precedence over others. Statecharts has become popular among engineers as a design notation for *embedded systems*, and commercially available tools provide support for it [10]. Nevertheless, precisely defining the semantics of the language has proved extremely challenging, with a variety of proposals [8, 9, 18, 19, 21, 28, 29] being offered for several dialects [34] of the language. While the research results have yielded insight into different aspects of the notation, no definitive account has emerged. This has an obviously undesirable practical ramification; tool builders for Statecharts must resort to *ad hoc* decisions in their implementations of semantically-based tools, such as model checkers [16, 23], and this means that designs developed by engineers have a meaning that may vary from implementation to implementation.

The semantic subtlety of Statecharts arises from the language's capability for defining transitions whose enablement disables other transitions. A Statechart may react to an *event* by engaging in an enabled transition, thereby performing a so-called *micro step*, which may generate new events that may in turn trigger new transitions while disabling others. When this chain reaction comes to a halt, one execution step, a so-called *macro step*, is complete. Technically, the difficulty for defining an operational semantics capturing the "macro-step" behavior of Statecharts arises from the fact that such a semantics should exhibit the following desirable properties: (i) the *synchrony hypothesis* [2], which guarantees that a reaction to an external event terminates before the next event enters the system, (ii) *compositionality*, which ensures that

*This work was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the first author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199, USA. The third author was supported by NSF grants CCR-9257963, CCR-9505662, CCR-9804091, and INT-9603441, AFOSR grant F49620-95-1-0508, and ARO grant P-38682-MA.

†ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23681-2199, USA, e-mail: luettgen@icase.edu.

‡Department of Computer Science, Munich University of Technology, Arcisstr. 21, D-80290 München, Germany, e-mail: beeck@in.tum.de.

§Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400, USA, e-mail: rance@cs.sunysb.edu.

2. Statecharts. Statecharts is a specification language for *reactive systems* [27], i.e., concurrent systems which are characterized by their ongoing interaction with their *environment.* They subsume finite state machines whose transitions are labeled by pairs of events, where the first component is referred to as *trigger* and may include *negated events*, and the second component is referred to as *action.* Intuitively, if the environment offers the events in the trigger, but not the negated ones, then the transition is triggered and can be executed; it fires, thereby producing the events in the label's action. Concurrency is achieved by allowing complex Statecharts to be composed from more simple ones running in parallel, which may communicate via broadcasting events. Elementary, or *basic states* in Statecharts may also be hierarchically refined by injecting other Statecharts. Concurrency and hierarchy are especially important concepts, since they allow for *bottom-up* and *top-down* specifications of systems.

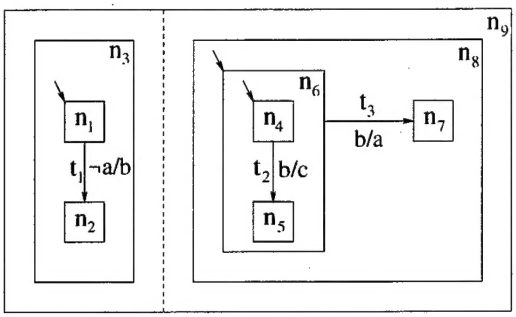


FIG. 2.1. *Example Statechart*

As an example, consider the Statechart depicted in Figure 2.1. It consists of a so-called *and-state*, labeled by n_9, which denotes the parallel composition of the two Statecharts labeled by n_3 and n_8. Actually, n_3 and n_8 are the names of *or-states*, describing sequential state machines. The first consists of two states n_1 and n_2 that are connected via transition t_1 with label $\neg a/b$. The label specifies that t_1 is triggered by $\neg a$, i.e., by the absence of event a, and produces event b. States n_1 and n_2 are not refined further and, therefore, are also referred to as basic states. Or-state n_8 is refined by or-state n_6 and basic state n_7, connected via a transition labeled by b/a. Or-state n_6 is further refined by basic states n_4 and n_5, and transition t_2 labeled by b/c.

It should be mentioned that the variant of Statecharts considered here does not include "features" present in some other variants. In particular, we prohibit *interlevel transitions*, i.e., transitions crossing borderlines of states, and triggers of the form in_n, where n is the name of a state. Moreover, state hierarchy does not impose implicit priorities to transitions, where transitions on higher levels of the hierarchy have precedence over transitions on lower levels; e.g., transition t_3 does not have priority over transition t_2 in our example. The impact of altering our approach to accommodate these concepts is discussed in Section 6.

2.1. Statecharts Terms. For our purposes, it is convenient to represent Statecharts not visually but by terms. This is also done in related work [17, 18, 31], and our approach closely follows the one described in [18]. Formally, let $\mathcal{N}$ be a countable set of names for Statecharts states, $\mathcal{T}$ be a countable set of names for Statecharts transitions, and Π be a countable set of Statecharts events. Moreover, we associate with every event $e \in \Pi$ its negated counterpart $\neg e$. We also lift negation to negated events by defining $\neg\neg e =_{\mathsf{df}} e$. Finally, we write $\neg E$ for $\{\neg e \mid e \in E\}$, if $E \subseteq \Pi \cup \{\neg e \mid e \in \Pi\}$. Then, the set of Statecharts terms is defined to be the least set satisfying the following rules.

TABLE 2.2
Step-construction function

function *step-construction*$(s,\ E)$; var $T := \emptyset$;
  while $T \subset \mathsf{enabled}(s, E, T)$ do choose $t \in \mathsf{enabled}(s, E, T) \setminus T$; $T := T \cup \{t\}$ od;
return T

TABLE 2.3
Function update

$\mathsf{update}([n], T') =_{\mathrm{df}} [n]$ $\quad$ $\mathsf{update}([n : \vec{s}], T') =_{\mathrm{df}} [n : (\mathsf{update}(s_1, T_1), \ldots, \mathsf{update}(s_k, T_k))]$

$$\mathsf{update}([n : \vec{s}; l; T], T') =_{\mathrm{df}} \begin{cases} [n : \vec{s}; l; T] & \text{if } T' = \emptyset \\ [n : (s_1, \ldots, \mathsf{update}(s_l, T'), \ldots, s_k); l; T] & \text{if } \emptyset \neq T' \subseteq \mathsf{trans}(s_l) \\ [n : (s_1, \ldots, \mathsf{default}(s_m), \ldots, s_k); m; T] & \text{if } \emptyset \neq T' = \{\langle t', l, E, A, m\rangle\} \subseteq T \\ [n] & \text{otherwise} \end{cases}$$

micro steps, or transitions, that are *triggered* by events offered by the environment or generated by other micro steps, that are mutually *consistent*, *compatible*, and *relevant*, and that obey *causality*. The Statecharts principle of *global consistency*, which prohibits an event to be present and absent in the same macro step, is subsumed by the notions of *triggered* and *compatible*.

A transition $t \in \mathsf{trans}(s)$ is *consistent* with $T \subseteq \mathsf{trans}(s)$, in signs $t \in \mathsf{consistent}(s, T)$, if t is not in the same parallel component as any transition in T. Formally,

$$\mathsf{consistent}(s, T) =_{\mathrm{df}} \{t \in \mathsf{trans}(s) \mid \forall t' \in T.\, t \perp_s t'\}. \tag{2.1}$$

Here, we write $t \perp_s t'$, if $t = t'$, or if there exists an and-state $[n : (s_1, \ldots, s_k)]$ in s, i.e., $n \in \mathsf{states}(s)$, such that $t \in \mathsf{trans}(s_i)$ and $t' \in \mathsf{trans}(s_j)$ for some $1 \leq i, j \leq k$ satisfying $i \neq j$.

A transition $t \in \mathsf{trans}(s)$ is *compatible* to all transitions in $T \subseteq \mathsf{trans}(s)$, in signs $t \in \mathsf{compatible}(s, T)$, if no event produced by t appears negated in a trigger of a transition in T. Formally,

$$\mathsf{compatible}(s, T) =_{\mathrm{df}} \{t \in \mathsf{trans}(s) \mid \forall t' \in T.\, \mathsf{act}(t) \cap \neg\mathsf{ev}(t') = \emptyset\} \tag{2.2}$$

A transition $t \in \mathsf{trans}(s)$ is *relevant* for s, in signs $t \in \mathsf{relevant}(s)$, if the root of the source state of t is in the configuration of s. Formally,

$$\mathsf{relevant}(s) =_{\mathrm{df}} \{t \in \mathsf{trans}(s) \mid \mathsf{root}(\mathsf{out}(t)) \in \mathsf{config}(s)\} \tag{2.3}$$

A transition $t \in \mathsf{trans}(s)$ is *triggered* by a set E of events, in signs $t \in \mathsf{triggered}(s, E)$, if the positive, but not the negative, trigger events of t are in E. Formally,

$$\mathsf{triggered}(s, E) =_{\mathrm{df}} \{t \in \mathsf{trans}(s) \mid \mathsf{ev}(t) \cap \Pi \subseteq E \text{ and } \neg(\mathsf{ev}(t) \cap \neg\Pi) \cap E = \emptyset\} \tag{2.4}$$

Finally, a transition t is *enabled* in configuration s regarding a set E of events and a set T of transitions, if $t \in \mathsf{enabled}(s, E, T)$, where

$$\mathsf{enabled}(s, E, T) =_{\mathrm{df}} \mathsf{relevant}(s) \cap \mathsf{consistent}(s, T) \cap \mathsf{triggered}(s, E \cup \bigcup_{t \in T} \mathsf{act}(t)) \cap \mathsf{compatible}(s, T) \tag{2.5}$$

and synchronization in *concurrent* systems. The role of actions in process algebras corresponds to the one of events in Statecharts. A clock represents the progress of time, which manifests itself in a recurrent global synchronization event, the clock transition, in which all process components are forced to take part. However, action and clock transitions are not orthogonal concepts that can be specified independently from each other, but are connected via the *maximal progress assumption* [11, 35]. Maximal progress implies that progress of time is determined by the *completion of internal computations* and, thus, mimics the synchrony hypothesis of Statecharts. The key idea for embedding Statecharts terms in a timed process algebra is to represent a macro step as a sequence of micro steps that is enclosed by clock transitions, signaling the beginning and the end of the macro step, respectively. This sequence implicitly encodes causality and, thus, leads to a compositional semantics for Statecharts, whose practicality does not suffer from complicated transition labels including causal orders [17, 18, 31].

Unfortunately, existing timed process algebras are, in their original form, not suitable for embedding Statecharts. The reason is that Statecharts transitions may be labeled by *multiple* events and that some events may appear in their *negated* form. The former feature implies that – in contrast to standard process algebras [1, 12, 24] – processes may be forced to synchronize on more than one event simultaneously, and the latter feature is similar to mechanisms for handling priority [4]. Moreover, our framework must include an operator similar to the *disabling operator* of LOTOS [3] for resembling state hierarchy [32]. Our Statecharts Process Language combines these well-known concepts in a single process algebra, which is expressive and flexible enough for embedding several Statecharts variants, as we will show below.

3.1. Syntax. Formally, let Λ be a countable set of *events* or *ports*, and let $\sigma \notin \Lambda$ be the distinguished *clock event* or *clock tick*. Based on Λ, we define *input actions* in SPL to be of the form $\langle E, N\rangle$, where $E, N \subseteq \Lambda$, and *output actions* E to be subsets of Λ. In case of the input action $\langle\emptyset,\emptyset\rangle$, we speak of an *unobservable* or *internal* action, which is also denoted by $\bullet$. Moreover, we let $\mathcal{A}$ stand for the set of all input actions. In contrast to CCS [24], the syntax of SPL includes two different operators for dealing with input and output actions, respectively. The *prefix operator* "$\langle E, N\rangle$." only permits prefixing with respect to input actions $\langle E, N\rangle$ which are instantly consumed in a single step. Output actions E are signaled to the environment of a process by attaching them to the process via the *signal* operator "$[E]\sigma(\cdot)$." They remain visible until the next clock tick σ occurs. The syntax of SPL is given by the following BNF

$$P \;::=\; \mathbf{0} \mid X \mid \langle E,N\rangle.P \mid [E]\sigma(P) \mid P+P \mid P \rhd P \mid P \rhd_\sigma P \mid P|P \mid P\setminus L$$

where $L \subseteq \Lambda$ is a *restriction set*, and X is a *process variable* taken from some countable domain $\mathcal{V}$. We also allow the definition of *equations* $X \stackrel{\text{def}}{=} P$, where variable X is assigned to term P. If X occurs as a subterm of P, we say that X is *recursively* defined. We adopt the usual definitions for *open* and *closed* terms and *guarded* recursion, and refer to the closed and guarded terms as *processes* [24]. The symbol $\mathcal{P}$ denotes the set of all processes and is ranged over by P and Q. Finally, the operators $\rhd$ and $\rhd_\sigma$ – called *disabling* and *enabling* operator, respectively – allow us to model state hierarchy.

3.2. Operational Semantics. The operational semantics of an SPL process $P \in \mathcal{P}$ is given by a *labeled transition system* $\langle \mathcal{P}, \mathcal{A}\cup\{\sigma\}, \longrightarrow, P\rangle$, where $\mathcal{P}$ is the set of states, $\mathcal{A}\cup\{\sigma\}$ the alphabet, $\longrightarrow \subseteq \mathcal{P}\times(\mathcal{A}\cup\{\sigma\})\times\mathcal{P}$ the transition relation, and P the start state. We refer to transitions with labels in $\mathcal{A}$ as *action transitions* and to those with label σ as *clock transitions*. For the sake of simplicity, we write $P \xrightarrow[N]{E} P'$ instead of $\langle P, \langle E, N\rangle, P'\rangle \in \longrightarrow$ and $P \xrightarrow{\sigma} P'$ instead of $\langle P, \sigma, P'\rangle \in \longrightarrow$. We say that P *may engage in a transition labeled by* $\langle E, N\rangle$ *or* σ*, respectively, and thereafter behave like process* P'. The transition relation

event in E and if all events in N are restricted. Finally, process variable X, where $X \stackrel{\text{def}}{=} P$, is identified with a process that behaves as a distinguished solution of the equation $X = P$.

TABLE 3.3
Operational semantics (clock transitions)

$$\textsf{tAct}\ \frac{-}{\langle E,N\rangle.P \xrightarrow{\sigma} \langle E,N\rangle.P}\ \langle E,N\rangle \neq \bullet \qquad \textsf{tOut}\ \frac{-}{[E]\sigma(P) \xrightarrow{\sigma} P} \qquad \textsf{tSum}\ \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P+Q \xrightarrow{\sigma} P'+Q'}$$

$$\textsf{tPar}\ \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P \mid Q \xrightarrow{\sigma} P' \mid Q'}\ \bullet \notin \mathrm{I}(P \mid Q) \qquad \textsf{tNil}\ \frac{-}{\mathbf{0} \xrightarrow{\sigma} \mathbf{0}} \qquad \textsf{tDis}\ \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P \rhd Q \xrightarrow{\sigma} P' \rhd Q'}$$

$$\textsf{tRes}\ \frac{P \xrightarrow{\sigma} P'}{P \setminus L \xrightarrow{\sigma} P' \setminus L}\ \bullet \notin \mathrm{I}(P \setminus L) \qquad \textsf{tRec}\ \frac{P \xrightarrow{\sigma} P'}{X \xrightarrow{\sigma} P'}\ X \stackrel{\text{def}}{=} P \qquad \textsf{tEn}\ \frac{P \xrightarrow{\sigma} P'}{P \rhd_\sigma Q \xrightarrow{\sigma} P' \rhd Q}$$

The operational rules for clock transitions deal with the maximal progress assumption, i.e., if $\bullet \in \mathrm{I}(P) =_{\text{df}} \{\langle E,N\rangle \mid \exists P'.\, P \xrightarrow[N]{E} P'\}$ then a clock tick σ is inhibited. The reason that transitions other than those labeled by $\bullet$ do not have pre-emptive power is that these only indicate the *potential* of progress, whereas $\bullet$ denotes *real* progress in our framework. Rule tNil states that inaction process $\mathbf{0}$ can idle forever. Similarly, process $\langle E,N\rangle.P$ may idle for clock σ, whenever $\langle E,N\rangle \neq \bullet$. The *signal operator* in process $[E]\sigma(P)$, which offers communications on the ports in E to its environment, disappears as soon as the next clock tick arrives and, thereby, enables process P. Time has to proceed equally on both sides of summation, parallel composition, and disabling, i.e., $P+Q$, $P \mid Q$, and $P \rhd Q$ can engage in a clock transition if and only if both P and Q can. The side condition of Rule tPar implements maximal progress and states that there is no pending communication between P and Q. The reason for the side condition in Rule tRes is that the restriction operator may turn observable input actions into the internal, unobservable input action $\bullet$ (see Rule Res) and, thereby, may pre-empt the considered clock transition. Finally, Rule tEn states that a clock tick switches the enabling to the disabling operator. Rule tRec does not require extra explanation.

The operational semantics for SPL possesses several pleasant algebraic properties which are known from timed process algebras [11, 35], such as (i) the *idling* property, i.e., $\bullet \notin \mathrm{I}(P)$ implies $\exists P' \in \mathcal{P}.\, P \xrightarrow{\sigma} P'$, for all $P \in \mathcal{P}$, (ii) the *maximal progress* property, i.e., $\exists P' \in \mathcal{P}.\, P \xrightarrow{\sigma} P'$ implies $\bullet \notin \mathrm{I}(P)$, for all $P \in \mathcal{P}$, and (iii) the *time determinacy* property, i.e., $P \xrightarrow{\sigma} P'$ and $P \xrightarrow{\sigma} P''$ implies $P' = P''$, for all $P, P', P'' \in \mathcal{P}$. Moreover, the summation and parallel operators are *associative* and *commutative*.

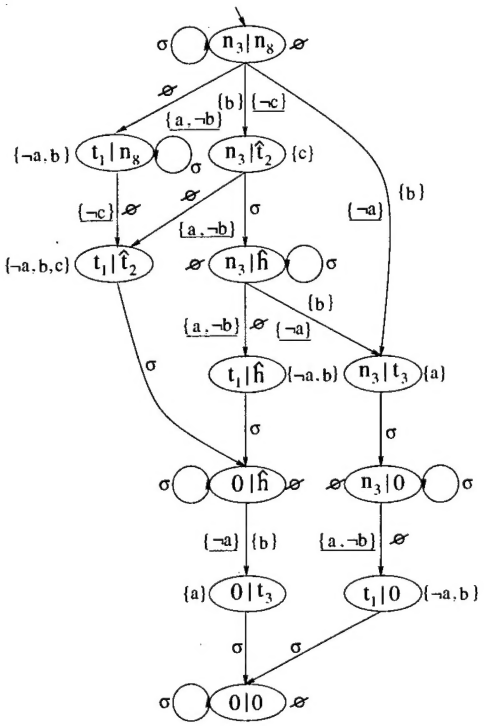
3.3. A Behavioral Equivalence. As shown above, the SPL operational semantics interprets processes as labeled transition systems. However, from a semantic point of view, several transition systems might describe the same observable system behavior. For coping with this situation, standard process algebras introduce *behavioral equivalences* which relate processes, or transition systems, that describe the same intuitive behavior. One popular behavioral equivalence is *bisimulation* [24] which may be adapted to cater for SPL as follows.

DEFINITION 3.1 (Bisimulation). Bisimulation equivalence, $\sim \subseteq \mathcal{P} \times \mathcal{P}$, *is the largest symmetric relation such that whenever* $P \sim Q$, *the following conditions hold.*

1. $\overline{\mathrm{I}}(P) \subseteq \overline{\mathrm{I}}(Q)$
2. *If* $P \xrightarrow[N]{E} P'$ *then* $\exists Q' \in \mathcal{P}.\, Q \xrightarrow[N]{E} Q'$ *and* $P' \sim Q'$.

TABLE 4.1
Embedding of the Example Statechart

$$
\begin{aligned}
[\![s_9]\!] &= n_9 \stackrel{\text{def}}{=} n_3 \,|\, n_8 \\
[\![s_3]\!] &= n_3 \stackrel{\text{def}}{=} \hat{n}_1 \\
\hat{n}_1 &\stackrel{\text{def}}{=} n_1 \rhd \langle \emptyset, \{a, \neg b\} \rangle . t_1 \\
t_1 &\stackrel{\text{def}}{=} [\{b, \neg a\}] \sigma(\hat{n}_2) \\
[\![s_1]\!] &= n_1 \stackrel{\text{def}}{=} \hat{n}_1 \stackrel{\text{def}}{=} \mathbf{0} \\
[\![s_2]\!] &= n_2 \stackrel{\text{def}}{=} \hat{n}_2 \stackrel{\text{def}}{=} \mathbf{0} \\
[\![s_8]\!] &= n_8 \stackrel{\text{def}}{=} \hat{n}_6 \\
\hat{n}_6 &\stackrel{\text{def}}{=} n_6 \rhd \langle \{b\}, \{\neg a\} \rangle . t_3 \\
t_3 &\stackrel{\text{def}}{=} [\{a\}] \sigma(\hat{n}_7) \\
[\![s_6]\!] &= n_6 \stackrel{\text{def}}{=} \hat{n}_4 \\
\hat{n}_4 &\stackrel{\text{def}}{=} n_4 \rhd \langle \{b\}, \{\neg c\} \rangle . t_2 \\
t_2 &\stackrel{\text{def}}{=} [\{c\}] \sigma(\hat{n}_5) \\
[\![s_4]\!] &= n_4 \stackrel{\text{def}}{=} \hat{n}_4 \stackrel{\text{def}}{=} \mathbf{0} \\
[\![s_5]\!] &= n_5 \stackrel{\text{def}}{=} \hat{n}_5 \stackrel{\text{def}}{=} \mathbf{0} \\
[\![s_7]\!] &= n_7 \stackrel{\text{def}}{=} \hat{n}_7 \stackrel{\text{def}}{=} \mathbf{0}
\end{aligned}
$$



been triggered. Accordingly, it offers the events in A until the current macro step is completed, i.e., until a clock transition is executed. In order to ensure global consistency, process t also offers the events in $E \cap \neg\Pi$. It is worth noting that SPL's two-level semantics of action and clock transitions allows for broadcasting events using SPL's synchronization mechanism together with its maximal progress assumption.

We now return to our introductory example by presenting its formal translation to SPL in Table 4.1, left-hand side. The embedding's operational semantics is depicted on the right-hand side of Table 4.1, where $\hat{t}_2 \stackrel{\text{def}}{=} t_2 \rhd_\sigma \langle \{b\}, \{\neg a\} \rangle . t_3$, and $\hat{h} \stackrel{\text{def}}{=} \mathbf{0} \rhd \langle \{b\}, \{\neg a\} \rangle . t_3$. Moreover, the initial output action set $\overline{\Pi}(P)$, for some $P \in \mathcal{P}$, is denoted next to the ellipse symbolizing state P, and the sets N' appearing in the label of transitions are underlined in order to distinguish them from the sets E'. Let us have a closer look at the leftmost path of the transition system, which passes the states $(n_3 \,|\, n_8)$, $(t_1 \,|\, n_8)$, $(t_1 \,|\, \hat{t}_2)$, $(\mathbf{0} \,|\, \hat{h})$, $(\mathbf{0} \,|\, t_3)$, and $(\mathbf{0} \,|\, \mathbf{0})$. The first three states are separated from the last three states by a clock transition. Hence, the considered sequence corresponds to two "potential" macro steps. We say "potential," since macro steps only emerge when composing our Statecharts embedding with an environment which triggers macro steps. The events needed to trigger the transitions and the actions produced by them can be extracted from a macro-step sequence as follows. For obtaining the trigger, consider all transition labels $\langle E, N \rangle$ occurring in the sequence, add up all events in components E, and include the negations of all positive events in components N. Regarding the generated actions, consider the set of positive events in the initial output action sets of the states preceding the clock transition which signals the end of the macro step. Thus, the first potential macro step of the example sequence is triggered by $\neg a$ and produces events b and c, whereas the second is triggered by b and produces a. The state names along a sequence also indicate the transitions which have fired. More precisely, whenever a state includes a variable $t \in \mathcal{T}$ at its top-level, transition t participates in the current macro step. Thus, for the first potential macro step, transitions t_1 and t_2 are chosen, whereas

corresponds to the firing of t_l in s. Vice versa, if $(\mathsf{Env}_E \mid [\![s]\!]) \setminus \Lambda$ is the origin of an SPL path to a process which can only engage in a clock transition to $(\mathbf{0} \mid P') \setminus \Lambda$ and which mimics the triggering of a transition sequence $T = (t_1, \ldots, t_k)$, then T can be generated by the step-construction function relative to s and E. Moreover, $[\![\mathsf{update}(s,T)]\!] \doteq P'$.

The formalization of the above intuition requires the following auxiliary properties, where $s \in \mathsf{SC}$ and $E, A \subseteq \Pi$. Here, T stands for an arbitrary prefix of the above transition sequence $(t_1, \ldots, t_k)$ interpreted as set, i.e., $T = \{t_1, \ldots, t_l\}$ for some $0 \leq l \leq k$, and $\mathsf{act}(T)$ stands for $\bigcup_{t \in T} \mathsf{act}(t)$.

1. $\exists t \in \mathsf{enabled}(s,E,A,T) \setminus T$ implies $[\![s,T]\!] \xrightarrow[N']{E'} P'$ for some $E', N' \subseteq \Lambda$ and $P' \in \mathcal{P}$, such that $P' \doteq [\![s, T \cup \{t\}]\!]$, $E' = (\mathsf{ev}(t) \cap \Pi) \setminus \mathsf{act}(T)$, and $N' = \neg(\mathsf{ev}(t) \cap \neg\Pi) \cup \neg\mathsf{act}(t)$.
2. $[\![s,T]\!] \xrightarrow[N']{E'} P'$ for some $E' \subseteq E$, $N' \cap (E \cup \neg A) = \emptyset$, and $P' \in \mathcal{P}$ implies $\exists t \in \mathcal{T}.\ P' \doteq [\![s, T \cup \{t\}]\!]$, $t \in \mathsf{enabled}(s,E,A,T) \setminus T$, $E' = (\mathsf{ev}(t) \cap \Pi) \setminus \mathsf{act}(T)$, and $N' = \neg(\mathsf{ev}(t) \cap \neg\Pi) \cup \neg\mathsf{act}(t)$.
3. $\mathsf{enabled}(s,E,A,T) \setminus T = \emptyset$ implies $[\![s,T]\!] \xrightarrow{\sigma} P'$ for some $P' \in \mathcal{P}$, where $P' \doteq [\![\mathsf{update}(s,T), \emptyset]\!]$, and $\forall \langle E', N' \rangle \in \mathrm{I}([\![s,T]\!])$. $E' \setminus E \neq \emptyset$ or $N' \cap (E \cup \neg A) \neq \emptyset$.
4. $[\![s,T]\!] \xrightarrow{\sigma} P'$ for some $P' \in \mathcal{P}$ and $E' \setminus E \neq \emptyset$ or $N' \cap (E \cup \neg A) \neq \emptyset$ for all $\langle E', N' \rangle \in \mathrm{I}([\![s,T]\!])$ implies $\mathsf{enabled}(s,E,A,T) \setminus T = \emptyset$ and $P' \doteq [\![\mathsf{update}(s,T), \emptyset]\!]$.

The above properties establish a micro-step level relationship between Statecharts terms and the processes occurring in their embedding. The proof of each property can be done by induction on the structure of s and uses our extensions of the enabled function (cf. Section 2.3) and the embedding mapping (cf. Section 4.2). □

5.2. Preservation Results. We close the technical part by returning to the behavioral relation $\sim$ of bisimulation equivalence. First, we state a preservation result involving $\sim$ and SPL's macro-step semantics.

THEOREM 5.3. *Let $P, P', Q \in \mathcal{P}$ such that $P \sim Q$ and $P \overset{E}{\underset{A}{\Longrightarrow}} P'$. Then $\exists Q' \in \mathcal{P}.\ Q \overset{E}{\underset{A}{\Longrightarrow}} Q'$ and $P' \sim Q'$.*

The validity of this theorem relies on the congruence property of $\sim$ for SPL. When combining the insights obtained by establishing Theorems 5.2 and 5.3, one may derive the following corollary which relates bisimulation equivalence and Statecharts macro-step semantics.

COROLLARY 5.4. *Let $E, A \subseteq \Pi$, $s \in \mathsf{SC}$, and $P \in \mathcal{P}$ such that $[\![s]\!] \sim P$. Then*

1. *$\forall s' \in \mathsf{SC}.\ s \overset{E}{\underset{A}{\Longrightarrow}} s'$ implies $\exists P' \in \mathcal{P}.\, P \overset{E}{\underset{A}{\Longrightarrow}} P'$ and $[\![s']\!] \sim P'$.*
2. *$\forall P' \in \mathcal{P}.\, P \overset{E}{\underset{A}{\Longrightarrow}} P'$ implies $\exists s' \in \mathsf{SC}.\, s \overset{E}{\underset{A}{\Longrightarrow}} s'$ and $[\![s']\!] \sim P'$.*

6. Adaptability to Other Statecharts Variants. For Statecharts, a variety of different semantics has been introduced in the literature. The comparison paper [34] surveys over twenty Statecharts variants. In this section, we show how our approach can be adapted to these variants and, thereby, testify to its flexibility. We focus on the most relevant issues of Statecharts semantics, which are identified in [34].

As is immanent in this paper, we favor an *operational semantics* over a *denotational* one, since we feel that operational models are more intuitive and, therefore, easier to understand. Moreover, operational models provide an immediate interface to verification tools which implement state-exploration techniques. An important observation of this paper is that the concept of a single, global clock together with maximal progress is the key to providing a *compositional, causal* state-machine semantics for Statecharts. Although the semantics is defined on the micro-step level, it allows for an easy identification of macro steps. The clock enforces global synchronizations which mark the beginning and end of macro steps. Thus, macro steps are represented as sequences of micro steps, which encode the underlying causality of Statecharts semantics.

compositionality holds on the micro-step level, i.e., the level of SPL action transitions, whereas responsiveness and causality is guaranteed on the macro-step level, i.e., the level on which sequences of SPL action transitions between global synchronizations, caused by clock ticks σ, are bundled together.

Uselton and Smolka [31] and Levi [17] also focused on achieving a clean, compositional semantics for Statecharts by referring to process algebras. In contrast to our approach, Uselton and Smolka's notion of transition system involves complex labels of the form $\langle E, \prec \rangle$, where E is a set of events and $\prec$ a transitive, irreflexive order on E, for encoding causality. Unfortunately, their semantics suffers from some serious problems, as pointed out in [17, 18]. Essentially, the semantics does not correspond – as intended – to the Statecharts semantics of Pnueli and Shalev [28]. Levi repaired this shortcoming by modifying the domains of the arguments of $\prec$ to sets of events and by allowing empty steps to be represented explicitly. However, we believe that our semantics, where labels do not contain any order at all, profits from improved readability.

Maggiolo-Schettini et al. considered a hierarchy of equivalences for Statecharts, including isomorphism and bisimulation, and studied congruence properties with respect to Statecharts operators [18]. For this purpose, they defined a compositional, operational macro-step semantics of Statecharts, which slightly differs from the one of Pnueli and Shalev since it does not allow the step-construction function to fail. In their semantics, labels of transitions consist of four-tuples which include information about causal orderings, global consistency, and negated events. This complexity prohibits an intuitive understanding of Statecharts semantics and an easy integration with existing analysis and verification tools. However, it should be noted that the semantic framework presented in [18] serves well for the purpose of studying certain algebraic properties of equivalences on Statecharts, such as fully-abstractness results and axiomatizations [14, 15].

Another popular design language with a visual appeal like Statecharts and, moreover, a solid algebraic foundation is *Argos* [20]. However, the semantics of Argos, defined via SOS rules as labeled transition systems, significantly differs from classical Statecharts semantics. For example, Argos is deterministic, abstracts from "non-causal" Statecharts by semantically identifying them with a *failure* state, and allows a single parallel component to fire more than once within a macro step.

Interfacing Statemate [10] to model-checking tools is a main objective in [16] and most recently also in a series of papers by Mikk et al. [21, 22, 23]. The first paper of this series includes a formalization of the semantics of Statemate, as defined in [8], within the specification formalism Z [30]. The second paper describes a translation from a subset of Statemate to *hierarchical state automata* which may be mapped to the specification language of the verification tool Spin [13], as shown in Mikk's third paper.

8. Conclusions and Future Work. This paper presented a process-algebraic approach to defining a compositional semantics for Statecharts. Our technique translates Statecharts terms to terms in the process algebra SPL which is expressive enough to model the semantic principles underlying Statecharts. SPL allows one to encode a "micro-step" semantics of Statecharts in the traditional SOS-style; it is at this level that our semantics is compositional, as bisimulation may be shown to be a congruence for the language. The macro-step semantics may then be given in terms of a derived transition relation. This semantics cannot be compositional, as results of Huizing and Gerth have shown [15]. However, the algebraic basis of our semantics permits the investigation of, e.g., the largest congruence consonant within this semantics. Also, since these sequences essentially encode total closures of causal orders, *partial order methods* might be useful for avoiding unnecessary state explosion in practice [6]. Note that, although SPL is a newly developed process algebra, all of its semantic ingredients have already been studied in the process-algebra community.

[11] M. HENNESSY AND T. REGAN, *A process algebra for timed systems*, Information and Computation, 117 (1995), pp. 221–239.

[12] C. HOARE, *Communicating Sequential Processes*, Prentice Hall, London, UK, 1985.

[13] G. HOLZMANN, *The model checker Spin*, IEEE Transactions on Software Engineering, 23 (1997), pp. 279–295.

[14] J. HOOMAN, S. RAMESH, AND W.-P. DE ROEVER, *A compositional axiomatization of Statecharts*, Theoretical Computer Science, 101 (1992), pp. 289–335.

[15] C. HUIZING, *Semantics of Reactive Systems: Comparison and Full Abstraction*, Ph.D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, March 1991.

[16] P. KELB, *Abstraktionstechniken für automatische Verifikationsmethoden*, Ph.D. thesis, University of Oldenburg, Oldenburg, Germany, 1996.

[17] F. LEVI, *Verification of Temporal and Real-Time Properties of Statecharts*, Ph.D. thesis, University of Pisa-Genova-Udine, Pisa, Italy, February 1997.

[18] A. MAGGIOLO-SCHETTINI, A. PERON, AND S. TINI, *Equivalences of Statecharts*, in Seventh International Conference on Concurrency Theory (CONCUR '96), U. Montanari and V. Sassone, eds., Vol. 1119 of Lecture Notes in Computer Science, Pisa, Italy, August 1996, Springer-Verlag, pp. 687–702.

[19] F. MARANINCHI, *The ARGOS language: Graphical representation of automata and description of reactive systems*, in IEEE Workshop on Visual Languages, IEEE Computer Society Press, October 1991.

[20] ——, *Operational and compositional semantics of synchronous automaton compositions*, in Third International Conference on Concurrency Theory (CONCUR '92), R. Cleaveland, ed., Vol. 630 of Lecture Notes in Computer Science, Stony Brook, NY, USA, August 1992, Springer-Verlag, pp. 550–564.

[21] E. MIKK, Y. LAKHNECH, C. PETERSOHN, AND M. SIEGEL, *On formal semantics of Statecharts as supported by STATEMATE*, in Second BCS-FACS Northern Formal Methods Workshop, Ilkley, UK, July 1997, Springer-Verlag.

[22] E. MIKK, Y. LAKHNECH, AND M. SIEGEL, *Hierarchical automata as model for Statecharts*, in Proceedings of Asian Computing Science Conference (ASIAN '97), Vol. 1345 of Lecture Notes in Computer Science, Springer-Verlag, December 1997.

[23] E. MIKK, Y. LAKHNECH, M. SIEGEL, AND G. HOLZMANN, *Verifying Statecharts with Spin*, in Proceedings of the Workshop on Industrial-Strength Formal Specification Techniques (WIFT '98), Boca Raton, FL, USA, October 1998, IEEE Computer Society Press.

[24] R. MILNER, *Communication and Concurrency*, Prentice Hall, London, UK, 1989.

[25] D. PARK, *Concurrency and automata on infinite sequences*, in Proceedings of 5th G.I. Conference on Theoretical Computer Science, P. Deussen, ed., Vol. 104 of Lecture Notes in Computer Science, Springer-Verlag, 1981, pp. 167–183.

[26] G. PLOTKIN, *A structural approach to operational semantics*, Tech. Report DAIMI-FN-19, Computer Science Department, Aarhus University, Denmark, 1981.

[27] A. PNUELI, ed., *Linear and Branching Structures in the Semantics and Logics of Reactive Systems*, Vol. 194 of Lecture Notes in Computer Science, Springer-Verlag, 1985.

[28] A. PNUELI AND M. SHALEV, *What is in a step: On the semantics of Statecharts*, in Theoretical Aspects of Computer Software (TACS '91), T. Ito and A. Meyer, eds., Vol. 526 of Lecture Notes in Computer Science, Sendai, Japan, September 1991, Springer-Verlag, pp. 244–264.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
October 1999

3. REPORT TYPE AND DATES COVERED
Contractor Report

4. TITLE AND SUBTITLE
Statecharts via process algebra

5. FUNDING NUMBERS
C NAS1-97046
WU 505-90-52-01

6. AUTHOR(S)
Gerald Lüttgen
Michael von der Beeck
Rance Cleaveland

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
ICASE Report No. 99-42

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-2199

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA/CR-1999-209713
ICASE Report No. 99-42

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Dennis M. Bushnell
Final Report
Presented at the International Conference on Concurrency Theory (CONCUR'99).

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category 60, 61
Distribution: Nonstandard
Availability: NASA-CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Statecharts is a visual language for specifying the behavior of reactive systems., The language extends finite-state machines with concepts of hierarchy, concurrency, and priority. Despite its popularity as a design notation for embedded systems, precisely defining its semantics has proved extremely challenging. In this paper, a simple process algebra, called Statecharts Process Language (SPL), is presented, which is expressive enough for encoding Statecharts in a structure-preserving and semantics-preserving manner. It is established that the behavioral relation bisimulation, when applied to SPL, preserves Statecharts semantics.

14. SUBJECT TERMS
bisimulation, compositionality, operational semantics, process algebra, statecharts

15. NUMBER OF PAGES
23

16. PRICE CODE
A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		